Design Patterns

Rev 1.2 (21/07/2019

Creational Patterns

Abstract Factory
Provide an interface for creating families of related or dependent objects without specifying their concrete class.

Builder
Separate the construction of a complex object from its representation allowing the same construction process to create various representations.

```
public Car construct() {
      return builder.setWheels(4)
                 .setColor("Red")
                 .build();
   }
```

Factory (method)
Define an interface for creating a single object but let subclasses decide which class to instantiate, factory lets a class defer instantiation to subclasses.

```
public class ShapeFactory {

   //use getShape method to get object of type shape
   public Shape getShape(String shapeType){
      if(shapeType == null){
         return null;
      }
      if(shapeType.equalsIgnoreCase("CIRCLE")){
         return new Circle();

      } else if(shapeType.equalsIgnoreCase("RECTANGLE")){
         return new Rectangle();

    ...
```

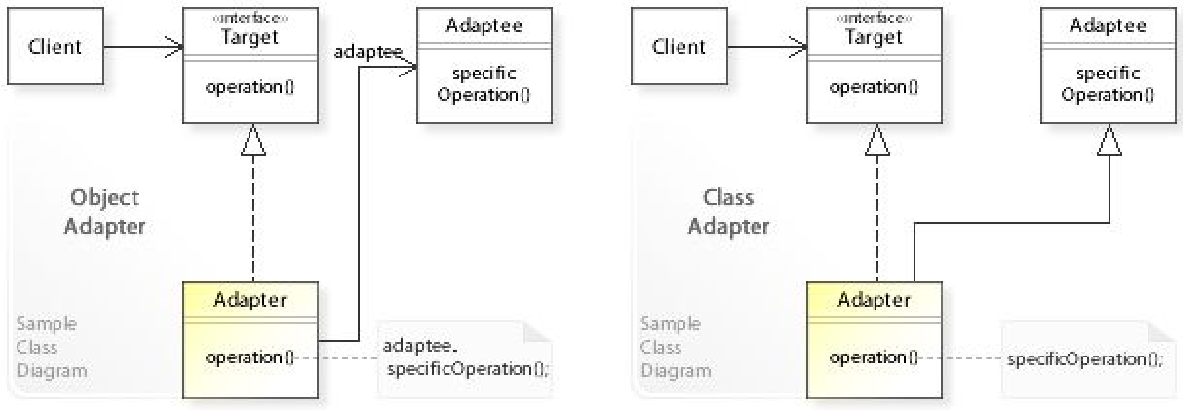
Virtual Proxy/Lazy initialization
Strategy of delaying the creation of an object, calculation of value or some expensive process until the first time it's needed.

Prototype
Specify the kinds of object to create by using a prototypical instance and create new objects from the skeleton of an existing object.
E.g. creation by cloning

Singleton
Ensure a class had only one instance and provide a global point of access to it.

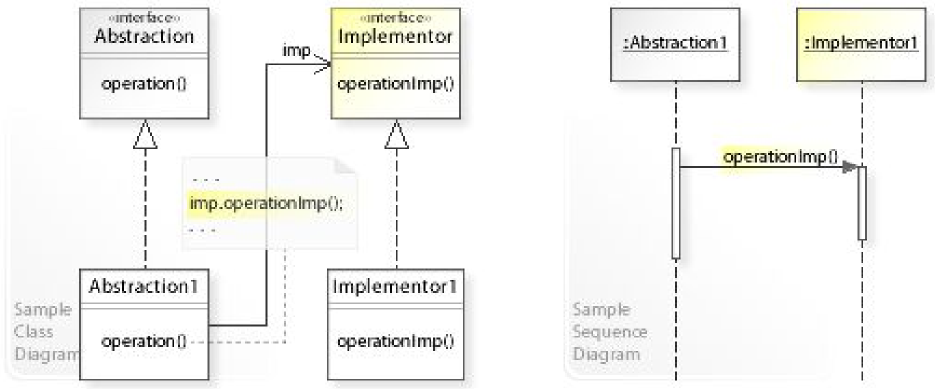
Structural Patterns

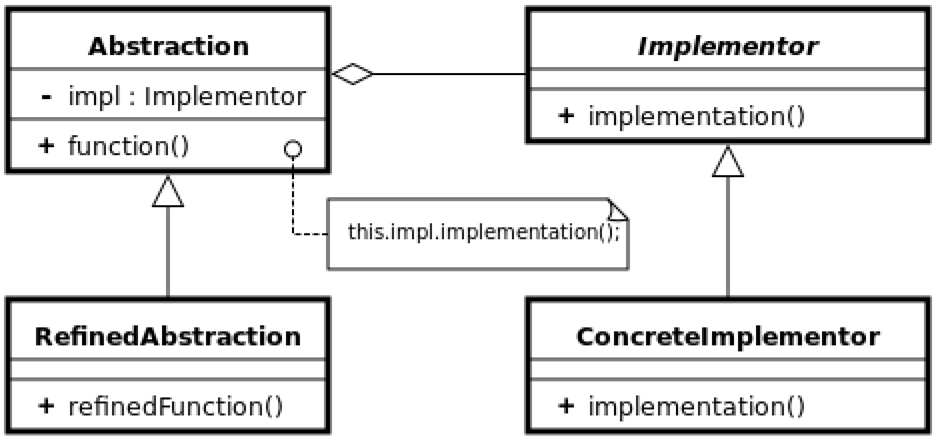
Adapter (Wrapper)
Adapter pattern works as a bridge between two incompatible interfaces, an adapter allows to classes to interact together that could not otherwise because of incompatible interfaces.



(Adapter inherits Target and contains adapter)

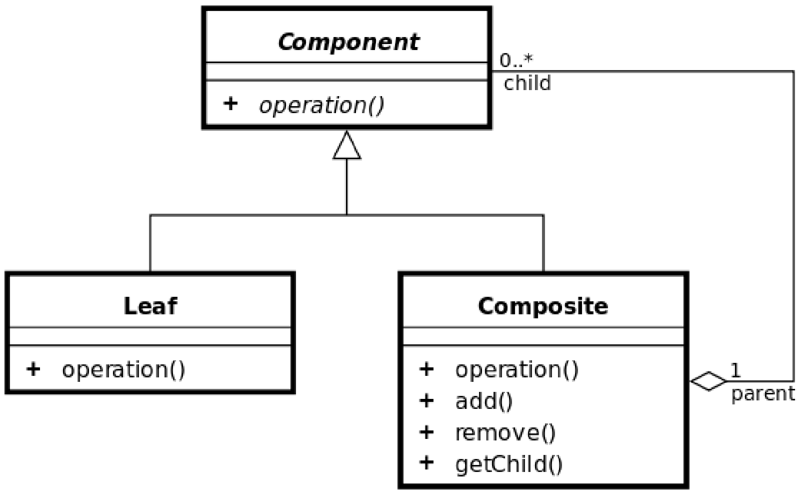
Bridge
Decouples an abstraction from its implementation allowing the to to vary independently.



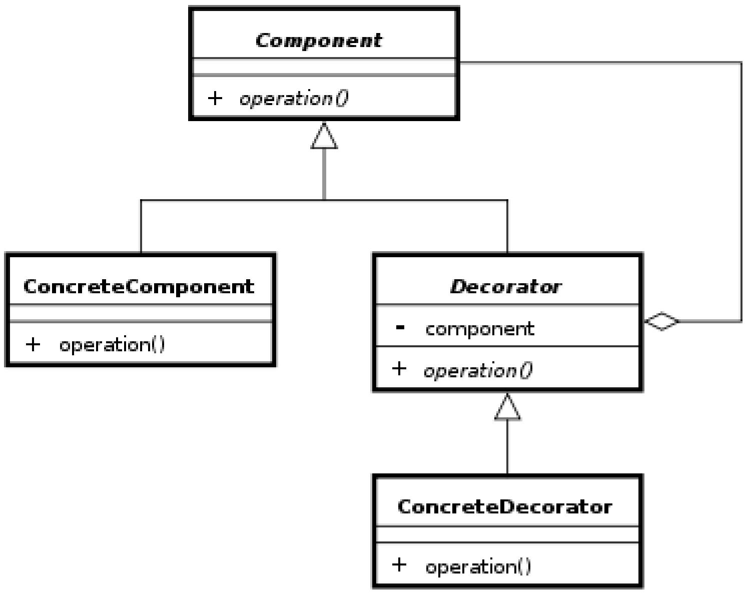


Abstraction (abstract class), defines the abstract interface, maintains the Implementor reference.
RefinedAbstraction (normal class).
Implementor (interface), defines the interface for implementation classes.
ConcreteImplementor (normal class), implements the Implementor interface.

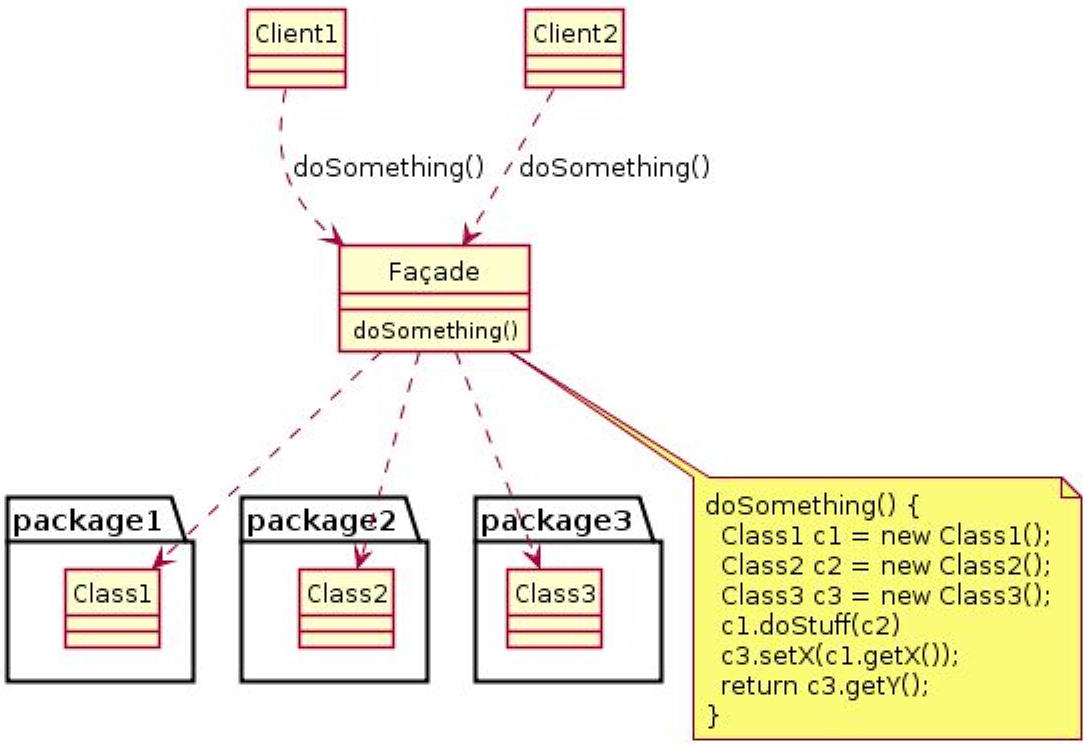
Composite
Compose objects into tree structures to represent part-whole hierarchies, composite allows individual objects and composition of objects to be treated uniformly.



Decorator
Attach additional responsibilities to an object dynamically keeping the same interface.



Facade
Provide an unified interface to a set of interfaces in a subsystem, defined a higher-level interface that makes the subsystem easier to use.



```
doSomething() {
  Class1 c1 = new Class1();
  Class2 c2 = new Class2();
  Class3 c3 = new Class3();
  c1.doStuff(c2)
  c3.setX(c1.getX());
  return c3.getY();
}
```

Flyweight
Uses sharing to support large number of similar objects efficiently.

Proxy
Provide a surrogate or placeholder for another object to control access to it.

Behavioral Patterns

Chain of Responsibility
Avoid coupling the sender of a request to its receiver by giving more than one object a change to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

E.g. try/catch block

Command
Encapsulate a request as an object, allowing for the parameterization of clients with different request and the queuing or logging of request.

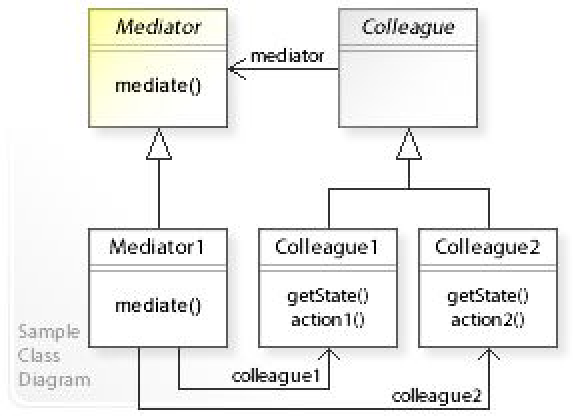
Interpreter
Given a language define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

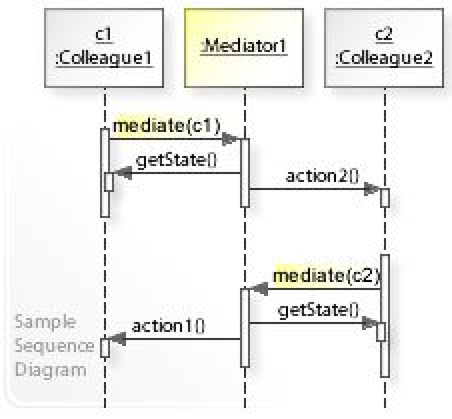
Iterator
Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

Memento
Without violating encapsulation capture and externalize an object internal state allowing the object to be restored later.

E. g. Persistence

Mediator
Define an object that encapsulates how a set of objects interact. Mediators promotes loose coupling by keeping objects from referring each other explicitly.

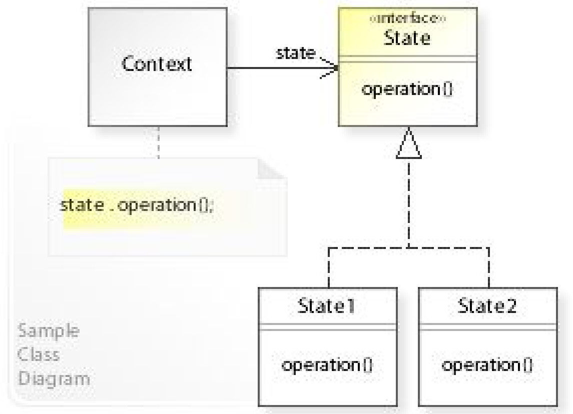


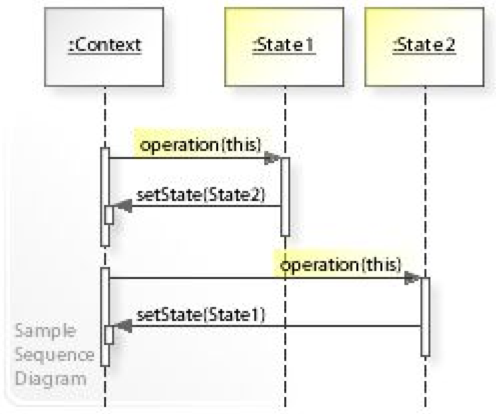


Observer
Define a one-to-many dependency between objects where a state change in one objects results in all its dependents being notified and updated automatically.

E.g. events

State
Allow an object to alter its behaviour when its internal state changes. The object will appear to change its class.

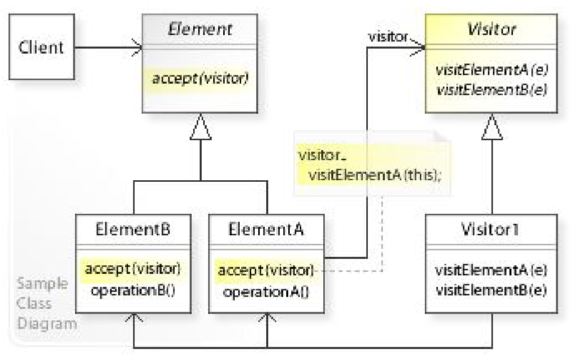


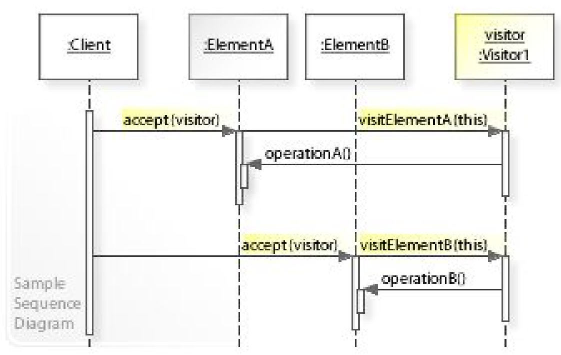


Strategy
Define a family of algorithms, encapsulate each one and make then interchangeable, strategy lets the algorithm vary independently from clients that use it.

Template
Define the skeleton of an algorithm in an operation deferring some steps to subclasses. Template methods lets subclasses redefine certain steps of an algorithm without changing its structure.

Visitor
Represent an operation to be performed on the elements of an object structure. Visitor lets a new operation to be defined without changing the classes of the elements on which operates.





E.g. lambda

Concurrency Patterns

Active Object
Decouples methods execution from method invocation that resides in their own thread of control. The goal is to introduce concurrency by using asynchronous method invocation and schedules for handling request.

Double-checked locking
Reduce the overhead of acquiring a lock by first testing the locking criteria (lock hint) in unsafe manner and only if it succeeds. Does the actual locking process.

Monitor object
An object whose methods are subject to mutual exclusion, this prevents multiple objects from erroneously trying to use it at the same time.

Reactor
A reactor is an object that provides an asynchronous interface to resources that must be handled asynchronously.